Jin-Soo Kim
(*jinsoo.kim@snu.ac.kr*)

Systems Software &
Architecture Lab.

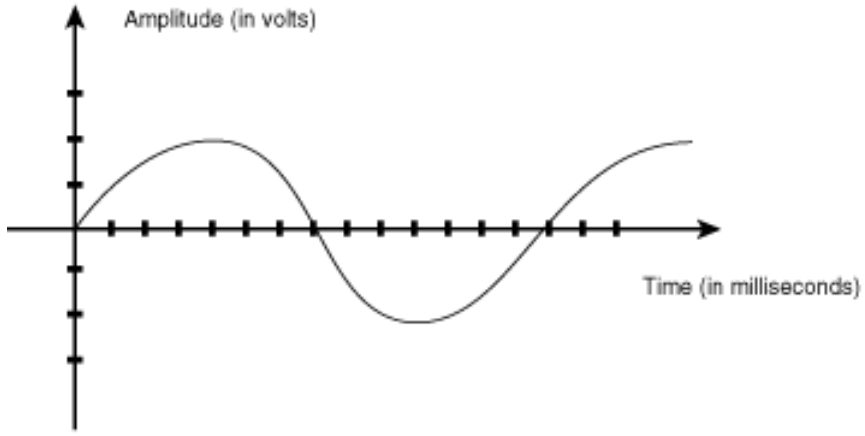
Seoul National University

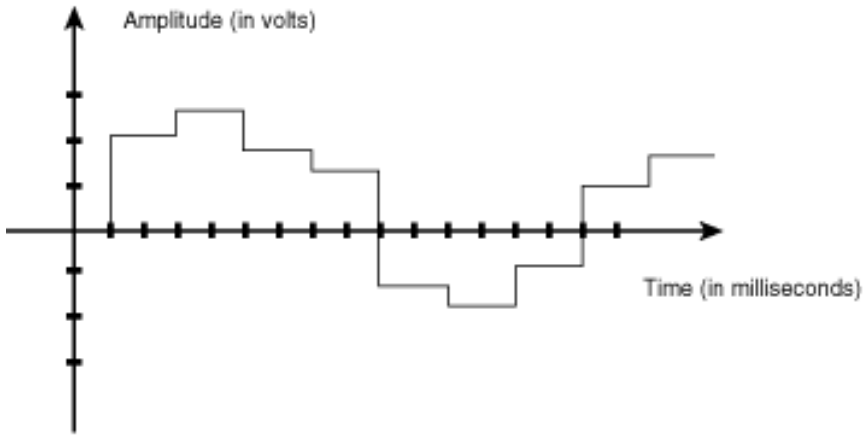
Fall 2022

Integers

The Advent of the Digital Age

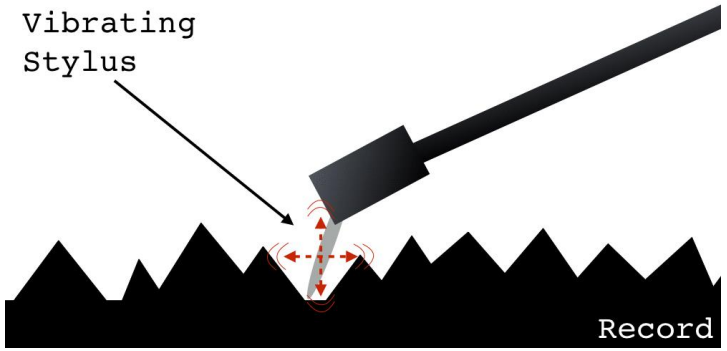
- Analog vs. digital?

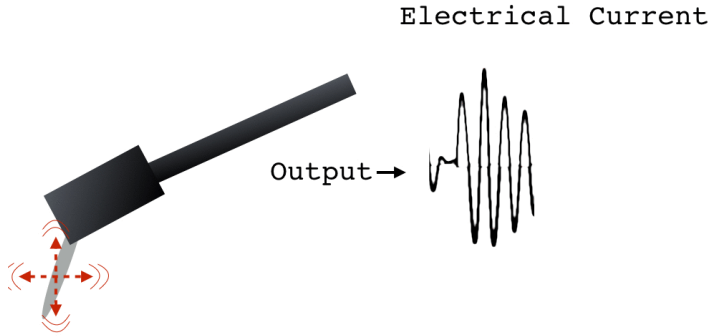


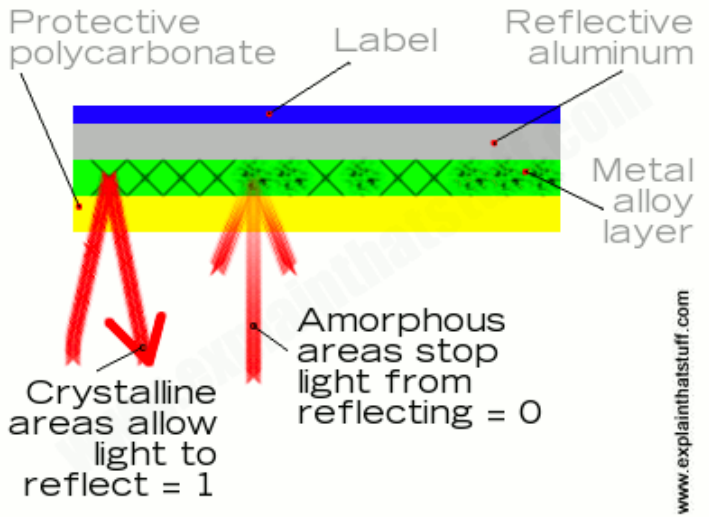


- Compact Disc (CD)
  - 44.1 KHz, 16-bit, 2-channel
- MP3
  - A digital audio encoding with lossy data compression

LP Record vs. Compact Disc

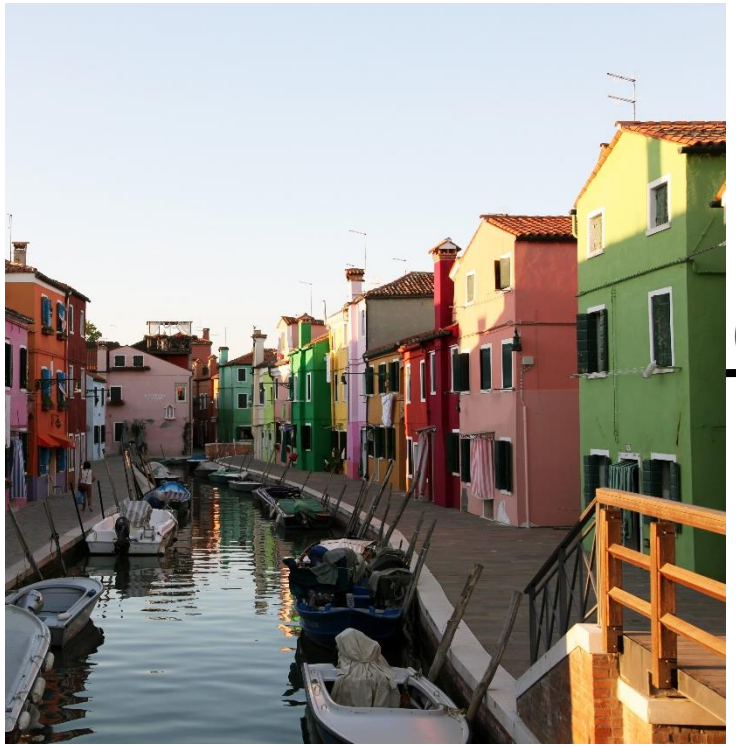


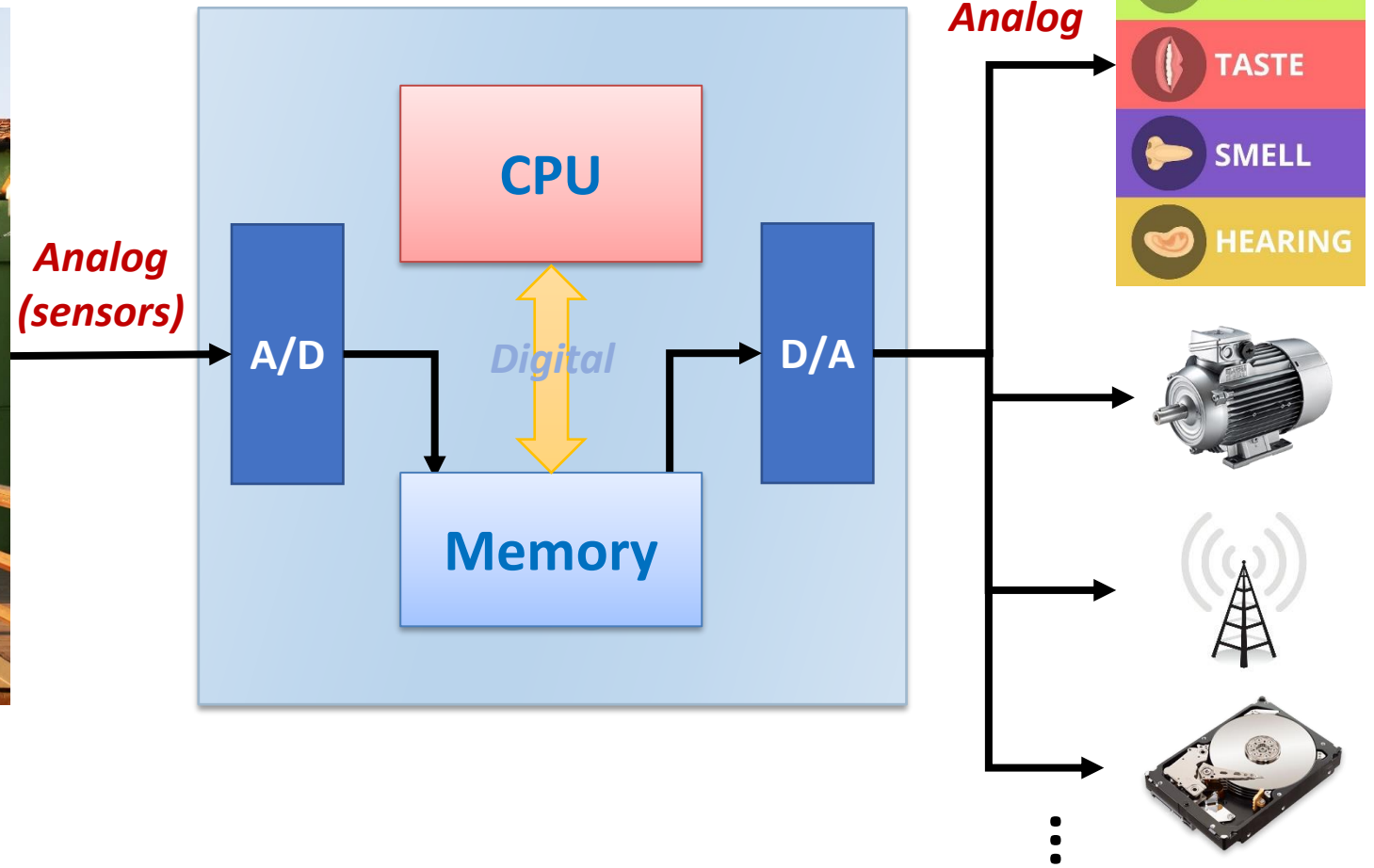




Source: http://www.soundsetal.com/blog-how-do-vinyl-records-work/
http://www.explainthatstuff.com/cdplayers.html

Digital Computer

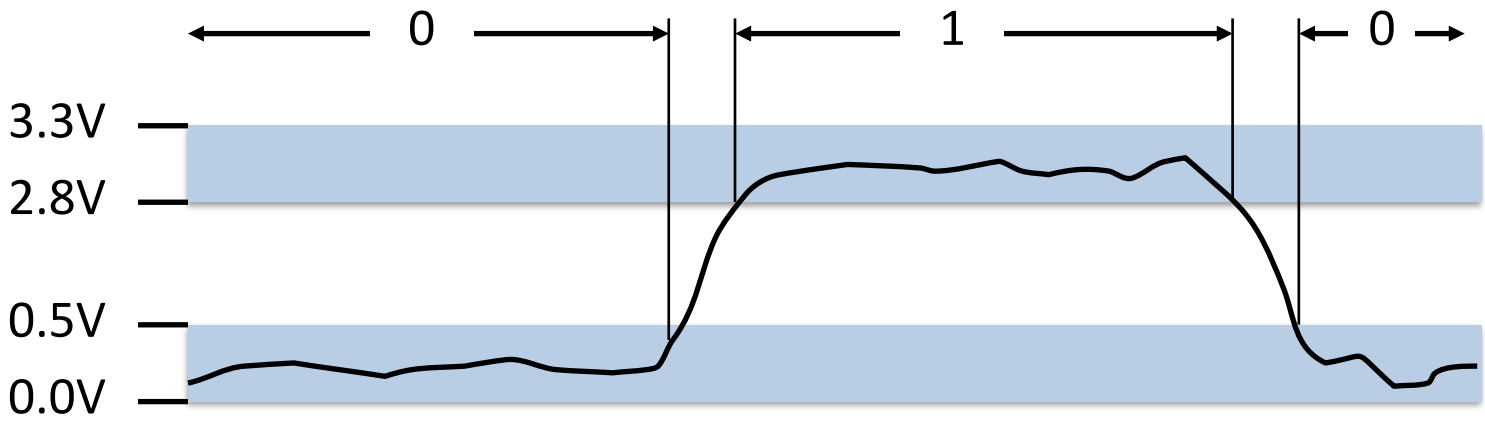




Binary Representations

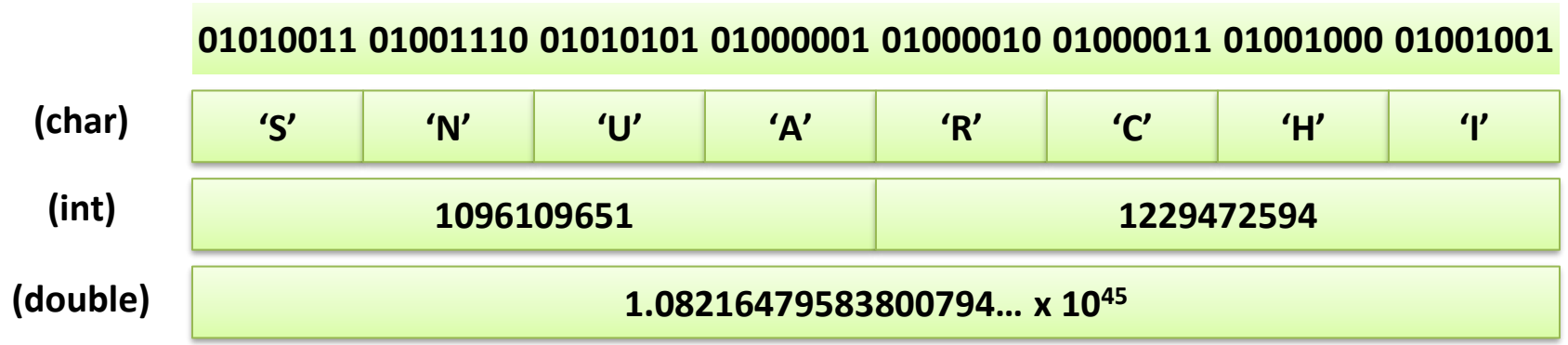
- Why not base 10 representation?
  - Easy to store with bistable elements
  - Straightforward implementation of arithmetic functions
  - Reliably transmitted on noisy and inaccurate wires

- Electronic implementation



Representing Information

- Information = Bits + Context
  - Computers manipulate representations of things
  - Things are represented as binary digits
  - What can you represent with *N* bits?
    - 2^N things
    - Numbers, characters, pixels, positions, source code, executable files, machine instructions, …
    - Depends on what operations you do on them



Encoding Byte Values

- Binary: 00000000_2 to 11111111_2
- Octal: 000_8 to 377_8
  - An integer constant that begins with `0` is an octal number in C
- Decimal: 0_{10} to 255_{10}
  - First digit must not be `0` in C
- Hexadecimal: 00_{16} to FF_{16}
  - Base 16 number representation
  - Use characters '0' to '9' and 'A' to 'F'
  - Write $FA1D37B_{16}$ in C as **`0xFA1D37B`** or **`0xfa1d37b`**

Hex	Decimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Representing Integers

Chap. 2.4

Unsigned Integers

- Encoding unsigned integers

$$B = [b_{w-1}, b_{w-2}, \ldots, b_0]$$

$$D(B) = \sum_{i=0}^{w-1} b_i \cdot 2^i$$

$x = 0001\ 0000\ 0101\ 1110_2$

$$D(x) = 2^{12} + 2^6 + 2^4 + 2^3 + 2^2 + 2^1$$
$$= 4096 + 64 + 16 + 8 + 4 + 2$$
$$= 4190$$

- What is the range for unsigned values with *w* bits?
- Using 64 bits: 0 to +18,446,774,073,709,551,615

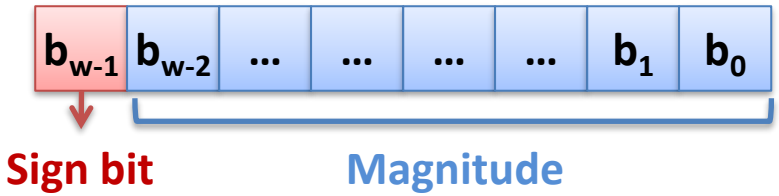
Signed Integers

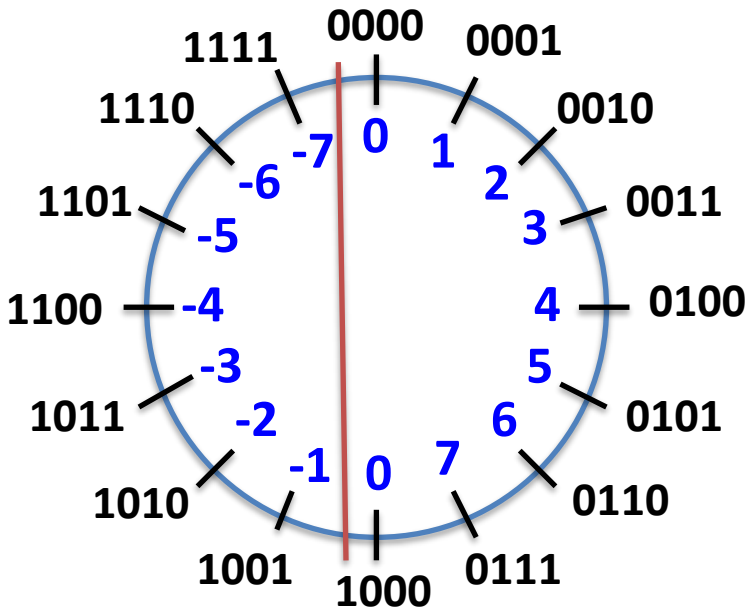
- Encoding positive numbers
  - Same as unsigned numbers

- Encoding negative numbers
  - Sign-magnitude representation
  - Ones' complement representation
  - Two's complement representation

Sign-magnitude Representation

- Two zeros
  - [000…00], [100..00]
- Used for floating-point numbers

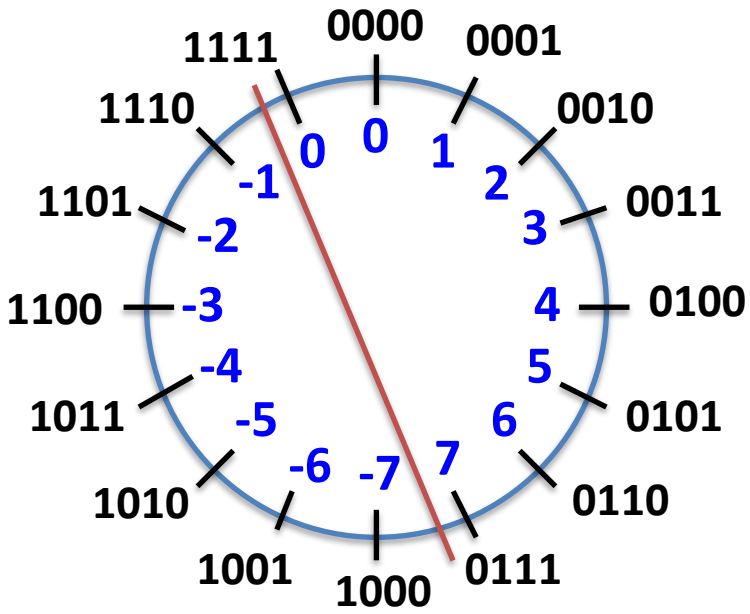




$$S(B) = (-1)^{b_{w-1}} \cdot \left(\sum_{i=0}^{w-2} b_i \cdot 2^i \right)$$

Ones' Complement Representation

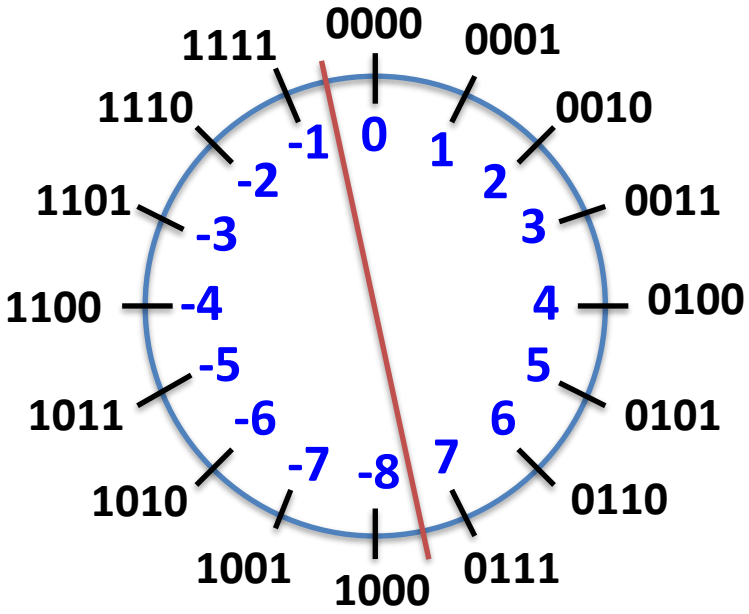
- Easy to find –*n*
- Two zeros
  - [000..00], [111..11]
- No longer used



$$O(B) = -b_{w-1}(2^{w-1} - 1) + \left(\sum_{i=0}^{w-2} b_i \cdot 2^i \right)$$

Two's Complement Representation (1)

- Unique zero
- Easy for hardware
  - leading 0 ≥ 0
  - leading 1 < 0
- Used by almost all modern machines



$$O(B) = -b_{w-1} \cdot 2^{w-1} + \left(\sum_{i=0}^{w-2} b_i \cdot 2^i \right)$$

Asymmetric range:

$$-2^{n-1} \sim 2^{n-1} - 1$$

Two's Complement Representation (2)

- Following holds for two's complement

$$\sim x + 1 == -x$$

- Complement
  - Observation: $\sim x + x == 1111...11_2 == -1$

- Increment

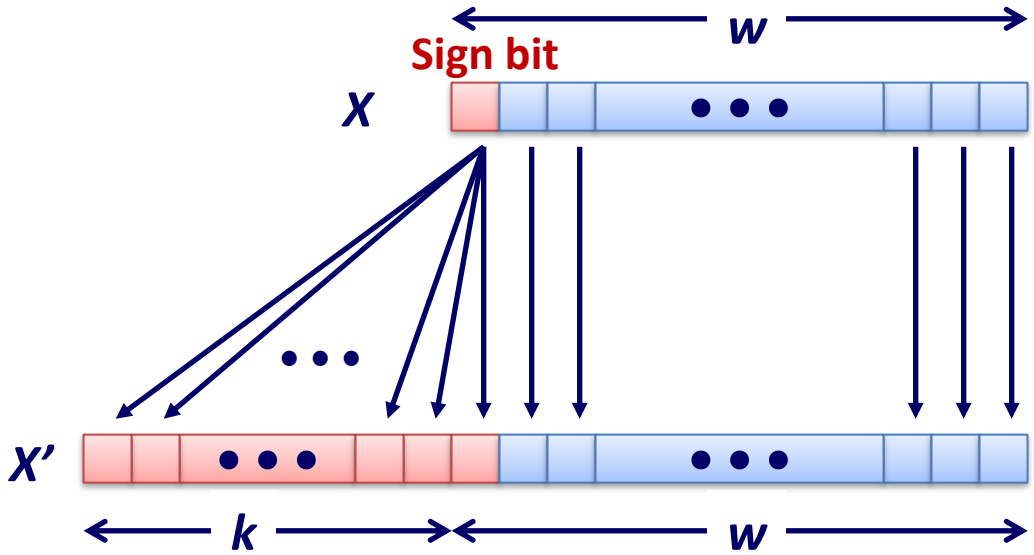
$\sim x + x == -1$

$\sim x + x + (-x + 1) == -1 + (-x + 1)$

$\sim x + 1 == -x$

Sign Extension

- Signed: ***w*** bits → ***w+k*** bits
  - Given ***w***-bit signed integer ***x***
  - Convert it to ***w+k***-bit integer with same value
- Replicate the sign bit to the left
  - +2: 0000 0010 → 0000 0000 0000 0010
  - -2: 1111 1110 → 1111 1111 1111 1110
- In RISC-V instruction set
  - `lb`: sign-extend loaded byte
  - `lbu`: zero-extend loaded byte



Manipulating Integers

Chap. 2.4, 3.1, 3.6

Bit-Level Operations in C

- Operations &, |, ~, ^ available in C
  - Apply to any "integral" data type
    - `(unsigned) long, int, short, char`
  - View arguments as bit vectors
  - Arguments applied bit-wise
- Examples (`char` data type)

`~0x41 → 0xBE`	$\sim 01000001_2 \rightarrow 10111110_2$
`~0x00 → 0xFF`	$\sim 00000000_2 \rightarrow 11111111_2$
`0x69 & 0x55 → 0x41`	$01101001_2 \;\&\; 01010101_2 \rightarrow 01000001_2$
`0x69 | 0x55 → 0x7D`	$01101001_2 \;\&\; 01010101_2 \rightarrow 01111101_2$
`0x69 ^ 0x55 → 0x3C`	$01101001_2 \;\&\; 01010101_2 \rightarrow 00111100_2$

Logic Operations in C

- &&, ||, !
  - View 0 as "False", anything nonzero as "True"
  - Always return 0 or 1
  - Early termination
- Examples (char data type)

```
!0x41     →  0x00
!0x00     →  0x01
!!0x41    →  0x01
0x69 && 0x55   →  0x01
0x69 || 0x55   →  0x01
if (p && *p) …                  // avoids null pointer access
```

Shift Operations

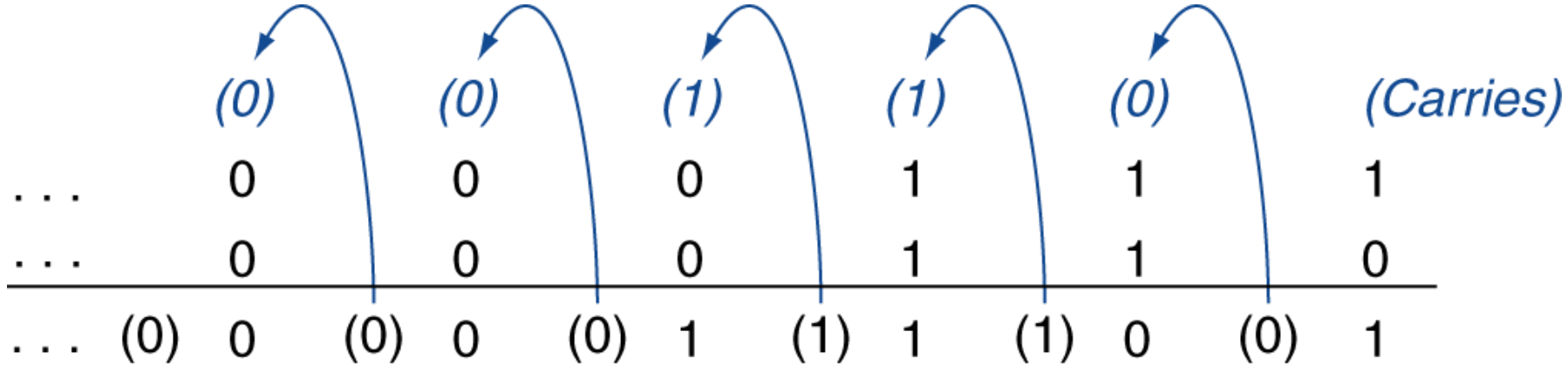
- Left shift: ***x << y***
  - Shift bit-vector ***x*** left ***y*** positions
    - Throw away extra bits on left
    - Fill with ***0***'s on right
- Right shift: ***x >> y***
  - Shift bit-vector ***x*** right ***y*** positions
    - Throw away extra bits on right
  - Logical shift: fill with ***0***'s on left
  - Arithmetic shift: replicate MSB on right
    - Useful with two's complement integer representation
- Undefined if ***y < 0*** or ***y*** ≥ word size

Argument x	01100010
<< 3	00010*000*
Log. >> 2	*00*011000
Arith. >> 2	*00*011000

Argument x	10100010
<< 3	00010*000*
Log. >> 2	*00*101000
Arith. >> 2	*11*101000

Addition

- Example: 7 + 6



- Overflow if result out of range
  - Adding +ve and –ve operands: No overflow
  - Adding two +ve operands: Overflow if result sign is 1
  - Adding two –ve operands: Overflow if result sign is 0

Addition: Signed vs. Unsigned

- Signed addition in C
  - Ignores carry output
  - The low-order ***w*** bits are identical to unsigned addition

Examples for 3-bit integer

Mode	x	y	x + y	Truncated x + y
Unsigned	4 [100]	3 [011]	7 [0111]	7 [111]
Two's comp.	-4 [100]	3 [011]	-1 [1111]	-1 [111]
Unsigned	4 [100]	7 [111]	11 [1011]	3 [011]
Two's comp.	-4 [100]	-1 [111]	-5 [1011]	3 [011]
Unsigned	3 [011]	3 [011]	6 [0110]	6 [110]
Two's comp.	3 [011]	3 [011]	6 [0110]	-2 [110]

Multiplication: Signed vs. Unsigned

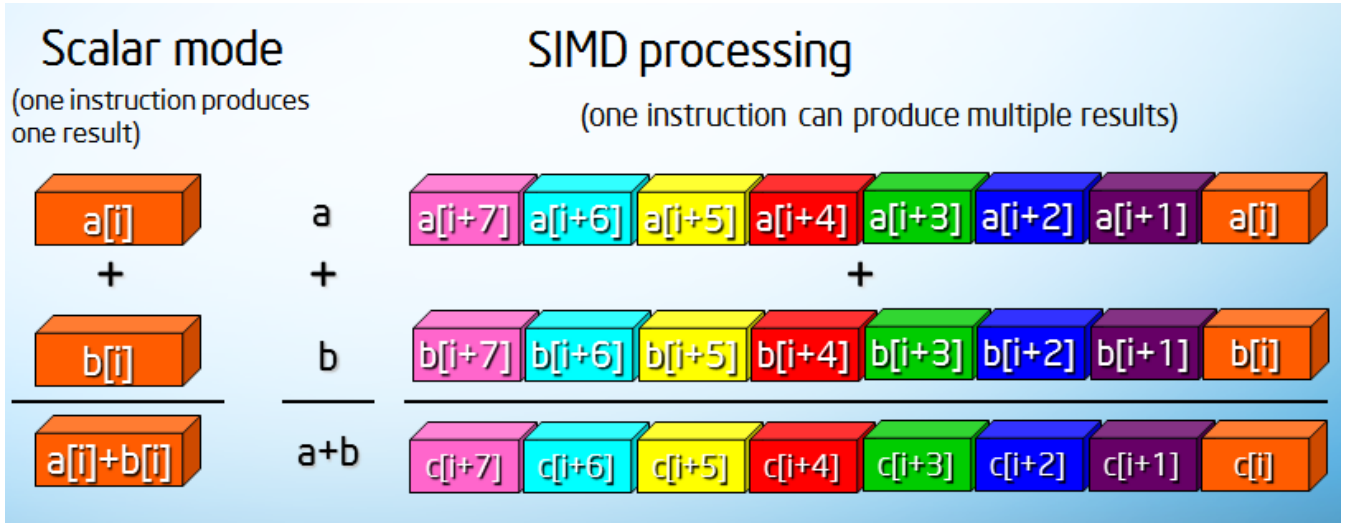
- Signed multiplication in C
  - Ignores high order ***w*** bits
  - The low-order ***w*** bits are identical to unsigned multiplication

Examples for 3-bit integer

Mode	x	y	x · y	Truncated x · y
Unsigned	5 [101]	3 [011]	15 [001111]	7 [111]
Two's comp.	-3 [101]	3 [011]	-9 [110111]	-1 [111]
Unsigned	4 [100]	7 [111]	28 [011100]	4 [100]
Two's comp.	-4 [100]	-1 [111]	4 [000100]	-4 [100]
Unsigned	3 [011]	3 [011]	9 [001001]	1 [001]
Two's comp.	3 [011]	3 [011]	9 [001001]	1 [001]

Arithmetic for Multimedia

- Graphics and media processing operates on vectors of 8-bit/16-bit data
  - Use 64-bit adder, with partitioned carry chain
  - Operate on 8x8-bit, 4x16-bit, or 2x32-bit vectors
  - SIMD (Single-Instruction, Multiple-Data)



- Saturating operations
  - On overflow, result is largest representable value
  - e.g., clipping in audio, saturation in video

Source: https://software.intel.com/en-us/blogs/2012/01/31/vectorization-find-out-what-it-is-find-out-more

Summary

- Two's complement for representing negative numbers
- Sign extension preserves the value
- Same hardware can be used for both signed and unsigned addition. This is also true for multiplication. But their overflow conditions are different.
- Remember?

```
int x = 50000;
printf ("%s\n", (x*x >= 0)? "Yes" : "No");
```